AN OPTIMAL DISTRIBUTED ALGORITHM FOR ALL-PAIRS SHORTEST-PATH

Saroja Kanchi and David Vineyard

Abstract: In this paper the network problem of determining all-pairs shortest-path is examined. A distributed algorithm which runs in O(n) time on a network of n nodes is presented. The number of messages of the algorithm is O(e+n log n) where e is the number of communication links of the network. We prove that this algorithm is time optimal.

Keywords: distributed algorithm, all-pairs shortest-path, computer network.

Introduction

In this paper we examine the distributed all-pairs shortest-path problem. The all-pairs shortest-path problem is the problem in which the shortest path between every pair of nodes in a network is determined. In the distributed version of the problem, a distributed algorithm is sought such that at the termination of the algorithm, every node knows the shortest path between any two nodes of the network. Floyd published a centralized algorithm [Floyd, 1962], which has been converted into a distributed algorithm by Toueg [Toueg, 1980]. The time complexity of this algorithm is $O(n^2)$ [Toueg, 1980].

The distributed shortest path problem and its variations have been studied because of its many applications. A decentralized algorithm for finding shortest paths in a network was presented by Abraham and Rhodes [Abram, 1978]. A distributed algorithm for finding shortest distances in an undirected graph was presented by Ravichandran, et. al. [Ravichandran, 1986] in which at the termination of the algorithm, each node contains the shortest path between itself and all other nodes. The algorithm described by Chandry and Misra [Chandry, 1982] finds shortest path from a node *i* to node *j* in a directed graph.

Determining topological properties of a network by distributed computation have received considerable attention. A number of papers have covered the topic of finding a minimum weight spanning tree [Awerbuch, 1987], [Korach, 1984], [Garay, 1998]. The problems of leader election, counting, and related problems [Awerbuch, 1987], [Singh, 1995], [Korach, 1984], [Kutten, 1998], [Kanchi, 1993] have also been studied.

In this paper we use the solution for finding a minimum weight spanning tree for finding a time optimal algorithm for the distributed all-pairs shortest-path problem.

There has been no previous distributed algorithm to find the all-pairs shortest-path in a general graph, other than the distributed version of a centralized algorithm given by Floyd [Floyd, 1962]. Therefore the idea of using a spanning tree and the center of the tree to find all-pairs shortest paths is a new element in this algorithm.

Model

The distributed network is considered to be an undirected weighted communications graph $G=(V,E)$, with processors forming the nodes, V, and bidirectional weighted communication links between processors forming the edges, E of the graph. No processor knows the topology of the network. No common memory is shared between processors and there is no global clock. All processors have unique identities from a totally ordered set. No processor knows the identity of any other processor. Each processor knows the links incident to it. For the duration of the algorithm, the network is assumed to be reliable, i.e., there will be no failures of the nodes or links for the duration of the algorithm.

The local computation at any node is assumed to take negligible time compared to the time required to transmit a message along a link. The asynchronous nature of the network permits undetermined communication delays in the delivery of a message. However, for the purpose of determining the time complexity, we assume that each message is delivered in $O(1)$ time along a link, irrespective of the size of the message. The correctness of the algorithm does not depend on this assumption.

The algorithm we present does not depend on any initiator node(s). At any time, one or more nodes may wake up and begin the execution of the algorithm. At the end of the algorithm, all nodes know the shortest path between any two nodes of the network. This data is stored in a square matrix, *D*, where entry *(i,j)* contains the shortest path between the nodes *i* and *j*.

Given any spanning tree *T* of a graph *G*, the edges that are not in *T* are called the *co-tree edges* with respect to *T*.

The size of the set *V* is denoted by *n*. The size of the set *E* is denoted by *e*.

Informal Description of the Algorithm

In this section we describe the algorithm at a high level. The algorithm consists of four steps as described below.

Step I: Finding a spanning tree, *T*, of the weighted graph, *G*:

Initially, all nodes are *inactive*. The first major part of the algorithm is to find a spanning tree, *T*, of the underlying unweighted graph. This can be accomplished by any one of the spanning tree finding algorithms, and we use the algorithm given by Awerbuch [Awerbuch, 1987], which takes time *O(n)*.

The spanning tree algorithm ensures every node can identify the links incident on it as either an edge in the tree *T* or a co-tree edge with respect to *T*.

Step II: Each node determines the identities of its neighbors in the graph *G*:

Each node must determine the identities of its neighbors in graph G. This can be accomplished by each node sending its identity along each link incident to it. The time complexity of this step is *O(1)*. Since each link carries exactly two messages, one from each of the incident nodes to the link, the number of messages is *2e*.

Step III: Determination of the All-Pairs Shortest-Distance matrix *D*:

This step of the algorithm deals with the transmission of distance information in *G* along the tree edges of *T*. Initially, each vertex constructs a local distance matrix that has row and column labels corresponding to the vertex and its neighbors.

Starting at each leaf node, partial distance information is transmitted along the tree edges of *T*. Whenever the partial distance matrix of a neighbor is received at a non-leaf node, new columns and rows are added to the partial distance matrix of that node and existing distance data is updated. When a non-leaf node receives partial distance matrix information from all but one of its neighbors, it becomes a transmitting node and sends its partial distance matrix to the neighbor from which it did not receive a partial distance matrix message.

At the end of this step, exactly one or two nodes, called the saturated node(s) of the tree, would contain Shortest-Distance matrix, *D*, of the entire graph *G*. We will show that the time complexity of this step is *O(n)*.

Step IV: Communicating the All-Pairs Shortest-Distance matrix *D* to every node:

This communication originates at the one or two nodes that are described in Step 3, and messages travel using only tree edges of *T*. This step has complexities of *O(n)* time and *O(n)* number of messages.

Notation Used in the Algorithm

Messages transmitted in this algorithm are of the following three types:

IDENTIFICATION: This type of message is used in Step II, where each node transmits its unique identity to each of its neighbors in the graph *G*.

PARTIAL DISTANCE MATRIX: This type of message in used in Step III, where the partial distance matrix calculated locally at a given node is sent along a single tree edge.

FINAL DISTANCE MATRIX: This type of message is used in Step IV, where the final distance matrix is sent to all the tree neighbors.

The nodes are in one of four states throughout the execution of the algorithm.

INACTIVE: Nodes are in Inactive state prior to the start of the algorithm. Initially all nodes are Inactive.

RECEIVING: Any non-leaf node that is receiving and processing partial distance matrices from other nodes is said to be in Receiving state. A node in Receiving state has not yet transmitted its partial distance matrix.

TRANSMITTING: A node is in Transmitting state if it has received partial distance matrices from all but one of its neighbors (this is trivially true for a leaf node). A node in Transmitting state sends its updated partial distance matrix to one other node from which it did not receive a partial distance matrix.

SATURATED: A node is in Saturated state if has received partial distance matrices from all its neighbors in the tree *T*.

Algorithm

In this section we describe the distributed algorithm for finding the all-pairs shortest-distance matrix.

ALGORITHM (ALL-PAIRS SHORTEST-PATH ALGORITHM)

1. Every node sets its state to Inactive.
2. Construct a spanning tree, *T* of the underlying unweighted graph. Any good asynchronous spanning tree algorithm can be used. The only modification to the spanning tree algorithm, which is required for our algorithm, is that any node with a single neighbor in the tree (a leaf node) must change its state to Transmitting at the end of the spanning tree algorithm. Similarly, any node with more than one neighbor in the tree (an interior node) must change its state to Receiving.
3. Each node *i* determines the identities of its neighbors in *G* and stores identity and distance data in a matrix PD_i. For instance, a node *i* that is adjacent to nodes *j* and *k*, creates entries *(i,j)*, *(j,k)* and *(i,k)* in PD_i. The value of $PD_i[i,j]$, $PD_i[i,k]$ would be the weights of the edges *(i,j)* and *(i,k)* respectively, and the value of $PD_i[j,k]$ would be the sum of weights of the edges *(i,j)* and *(i,k)*. See INITIALIZE_PARTIAL_DISTANCE_MATRIX subroutine below.
4. Determine All-Pairs Shortest Distance Matrix *D* of the graph *G*. Each node's behavior is determined by its state.

   For each node $i \in V$

   If the state of *i* is Receiving

   Run the subroutine RECEIVING_NODE_PROCESSING(*i*);

   If the state of *i* is Transmitting

   Run the subroutine TRANSMITTING_NODE_PROCESSING(*i*)

   As a result, at most 2 transmitting nodes will receive a message from all neighbors and are marked Saturated.
5. Transmit the final All-Pairs Shortest-Distance matrix to every node from a Saturated node. Any Saturated node contains the final all pairs shortest distance matrix *D*. The Saturated node(s) will create a final message consisting of *D* and send this message to all its neighbors in the spanning tree *T*. Any node in the spanning tree that receives *D* will store *D* locally and send *D* to all its tree neighbors except the tree neighbor from which it received *D*.

SUBROUTINE (INITIALIZE_PARTIAL_DISTANCE_MATRIX)

1. For each node $i \in V$
2. i transmits an Identification message containing its identity along each edge incident at i in G
3. i, upon receiving the identities of its m neighbors, creates a distance matrix, PD_i, of size $(m+1)\times(m+1)$ and assigns the values to $PD_i[j,k]$ as given below.
   3.1. For each j, $k \in$ indexes of PD_i
   3.2. If $j == k$ then $PD_i[j,k] \leftarrow 0$.
   3.3. If $j == i$ or $k == i$ then $PD_i[j,k] \leftarrow$ weight of the edge between j and k.
   3.4. Else $PD_i[j,k] \leftarrow PD_i[j,i] + PD_i[i,k]$.
   3.5. EndFor

SUBROUTINE (RECEIVING_NODE_PROCESSING(i))

1. Let $Tnbr_i$ be the set of neighbors of node i in Tree T created in Step 2 of the all-pairs shortest-path algorithm.
2. Let *count* be the number of the partial distance matrices that i has received since it changed state to Receiving. Initially count is set to 0.
3. Let Links_Used be a vector of size $|Tnbr_i|$ of type boolean in which all entries are initialized to False.
4. While count < $|Tnbr_i| - 1$
5. Receive message PD_j from neighbor j
6. count++
7. Link_Used[j] ←True
8. Call ProcessMessage(PD_j)
9. EndWhile
10. Set the state of node i to Transmitting.

SUBROUTINE (PROCESSMESSAGE(PD_j))

1. For each index k in PD_j
2. if k is not an index of PD_i
3. extend PD_i by one row and one column corresponding to k
4. For all indexes m in PD_i
5. Set $PD_i[k, m] \leftarrow PD_i[m, k] \leftarrow \infty$
6. EndFor
7. Set $PD_i[k, k] \leftarrow 0$
8. EndIf
9. EndFor
10. For each $k, m \in$ indexes of PD_j
11. if $PD_i[k,m] > PD_j[k,m]$
12. $PD_i[k,m] \leftarrow PD_j[k,m]$
13. EndFor
14. For each $k, m, n \in$ indexes of PD_i
15. if $PD_i[k,m] > PD_i[k,n] + PD_i[n,m]$
16. $PD_i[k,m] \leftarrow PD_i[k,n] + PD_i[n,m]$
17. EndFor

SUBROUTINE (TRANSMITTING_NODE_PROCESSING(i))

1. Node i transmits PD_i to its only neighbor in T from which it has not received a partial distance matrix.
2. If i receives another partial distance message, say from j, then i calls ProcessMessage(PD_j) and marks itself as Saturated.

Correctness

In this section we show that the All-Pairs Shortest-Path algorithm produces the correct result.

Lemma 1 There are at most two Saturated nodes.

PROOF: The algorithm starts at leaf nodes of the tree, and matrices are transmitted to internal nodes. Each internal node, in turn chooses the one node from which it has not received any partial distance matrix as its parent and transmits the partial distance matrix to that node. In this manner eventually the matrices reach the one or two centers of the tree. These centers become the Saturated nodes.

Lemma 2 The shortest distance between any two nodes is known to a Saturated node.

PROOF: We will prove this using induction on the number of edges in the shortest path. Any shortest path consisting of a single edge is known to the Saturated node(s), since every node, by Step 3, creates a partial distance matrix and all these matrices are transmitted eventually to the Saturated node(s).

Assume that if there are fewer than k edges in the shortest path between two nodes, then that path is known to the Saturated node(s). Consider two nodes x and y such that the shortest path P between x and y has k edges. Let $P = (x = v_0, v_1, v_2, ...,v_{k-1}, v_k = y)$. Assume that the Saturated node(s) contains a ``path" P' between x and y, but that the sum of the edge weights of P' is greater than the sum of the edge weights of P. Then the two paths must differ in at least one edge. Let (v_i, v_{i+1}) be the first edge in P that is not in P'. Note that v_i could be the same as x or v_{i+1} could be same as y. But since P is the shortest path from x to y, the path $(x, v_1, v_2, \dots , v_i)$ is a shortest path from x to v_i. Similarly, the path $(v_{i+1}, v_{i+2}, \dots,v_{k-1}, y)$ is a shortest path from v_{i+1} to y. Note that these paths must contain fewer than k edges, since P has k edges. But by the induction hypotheses the Saturated node contains the shortest path from x to v_i and from v_{i+1} to y since the number of edges in each of these shortest paths is less than k. Also, by Step 4 of the all-pairs shortest-path algorithm, Process_Message combines these two shortest paths to obtain the shortest path between x and y. Therefore the Saturated node must have the path P.

Theorem 1 The All-Pairs Shortest-Path Algorithm guarantees that all nodes in G know the all-pairs shortest-paths.

PROOF: By Lemma 1, there are exactly one or two Saturated nodes. By Lemma 2, a Saturated node knows the all-pairs shortest-path matrix D. Step 5 of the algorithm is a broadcast of this information to all nodes in the spanning tree, hence in the graph.

Complexity

In this section, we show that Algorithm 1 takes $O(n)$ time and $O(e + n \log n)$ number of messages. Note that the subroutines Initialize_Partial_Distance_Matrix, Receiving_Node_Processing(i), ProcessMessage, and Transmitting_Node_Processing each perform local processing and are thus considered to take $O(1)$ time.

Theorem 2 The all-pairs shortest-path algorithm terminates in $O(n)$ time.

PROOF: Step 1 of the algorithm takes $O(1)$ time. Step 2 of the algorithm, i.e., constructing the spanning tree, takes $O(n)$ time. [Awerbuch, 1987]. Step 3 of the algorithm takes $O(1)$ time, since each node sends one message on each tree link. Step 4 of the algorithm takes time proportional to the height of the tree with a Saturated node as a root. This is at most $O(n)$. Step 5 takes the same time as Step 4 since the messages travel from the root to the leaves of the tree. The time complexity of the algorithm is dominated by Step 2, and is thus $O(n)$.

Theorem 3 The all-pairs shortest-path algorithm has $O(e + n \log n)$ bound on the number of messages.

PROOF: The number of messages in Step 2 of the algorithm is $O(e + n \log n)$ [Awerbuch, 1987]. The number of messages in Step 3 of the algorithm is $2e$ since each edge is used for exactly two IDENTIFICATION messages. The number of messages in Step 4 of the algorithm is $O(n)$ because the partial distance matrices are transmitted from leaf nodes to the root of the tree (Saturated node) using only edges of T. The spanning tree has n-1 edges and

exactly one message is sent along each tree edge, thus the number of messages is *O(n)*. Note that if there are two Saturated nodes, the edge between them is used twice. Similarly, the number of messages in Step 5 of the algorithm is *O(n)*. Therefore the number of messages generated by the algorithm is bounded by

O(e +n log n).

Optimality

We claim that our distributed algorithm is time optimal for finding all-pairs shortest-path. This follows since a solution to the leader election problem can be obtained from a solution to the all-pairs shortest-path problem with no additional communication time. For instance, each node can locally elect the node with the highest identity as the leader. Since the time optimal leader election algorithm [Awerbuch, 1987] takes *O(n)* time, our *O(n)* time algorithm for all-pairs shortest-path is also time optimal.

Conclusion

We have developed a distributed algorithm for the all-pairs shortest-path problem which is optimal in time and number of messages. The optimal time is *O(n)*. The optimal number of messages is *O(e + n log n)*.

Bibliography

[Abram, 1978] J.M. Abram and I.B. Rhodes, *A decentralized shortest path algorithm* in Proc. of the 16th Allerton Conf. on Communication, Control, and Computing (Monticello, Ill.), pp. 271-277, 1978

[Awerbuch, 1987] B. Awerbuch, *Optimal distributed algorithms for minimum-weight spanning tree, counting, leader election and related problems*, in Proc. 19th ACM Symp. on Theory of Computing, ACM, New York, pp. 230-240, 1987

[Chandry, 1982] K.M. Chandry and J. Misra*, Distributed computation on graphs: shortest path algorithms*, Comm. ACM 25, pp. 833-837, Nov. 1982

[Floyd, 1962] R. Floyd, *Algorithm 97: shortest path*, Comm. ACM 5, 1962

[Garay, 1998] J. Garay, S. Kutten, and D. Peleg, *A sublinear time distributed algorithm for minimum-weight spanning trees*, SIAM J. Comput., Vol. 27, No. 1, pp. 302-316, February 1998

[Kanchi 1993] S.P. Kanchi and J.L. Kim, *Alternate algorithms for leader election on reliable and unreliable complete networks*, Proc. of the sixth international conf. on parallel and distributed computing and systems p.118-121, October 1993

[Korach, 1984] E. Korach, S. Moran, and S. Zaks, *Tight lower and upper bounds for some distributed algorithms for a complete network of processors*, Proc. of 1985 PODC Conf., Vancouver, BC, pp. 199-207, August 1984

[Kutten, 1998] S. Kutten and D. Peleg, *Fast distributed construction of small k-dominating sets and applications*, Journal of Algorithms 28, pp. 40-66, 1998

[Ravichandran, 1986] A. Ravichandran, S.G. Menon, and R.K. Shyamasundar, *A distributed algorithm for finding the shortest paths in an undirected graph*, Technical Report CS-86-13, Department of Computer Science, Pennsylvania State University, May 1986

[Singh, 1995] G. Singh and A. Bernstein, *A highly asynchronous minimum spanning tree protocol*, Distrib. Comput., pp 151-161, 1995

[Toueg, 1980] S. Toueg, *An all-pairs shortest-path distributed algorithm*, Res. Rep. RC-8327, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., 1980

Authors' Information

Saroja Kanchi – Department of Science and Mathematics, Kettering University, 1700 West Third Avenue, Flint, Michigan 48504-4898, USA: skanchi@kettering.edu

David Vineyard – Department of Science and Mathematics, Kettering University, 1700 West Third Avenue, Flint, Michigan 48504-4898, USA: dvineyar@kettering.edu